An Automatic Three-Dimensional Finite Element Mesh Generation System for the Poisson–Boltzmann Equation

CHRISTIAN M. CORTIS,[1]* RICHARD A. FRIESNER[2]

[1]*Department of Applied Physics and Center for Biomolecular Simulation, Columbia University, New York, New York 10027*

[2]*Department of Chemistry and Center for Biomolecular Simulation, Columbia University, New York, New York 10027*

Received 3 September 1996; accepted 12 March 1997

ABSTRACT: We present an automatic three-dimensional mesh generation system for the solution of the Poisson–Boltzmann equation using a finite element discretization. The different algorithms presented allow the construction of a tetrahedral mesh using a predetermined spatial distribution of vertices adapted to the geometry of the dielectric continuum solvent model. A constrained mesh generation strategy, based on Bowyer's algorithm, is used to construct the tetrahedral elements incrementally and embed the Richards surface of the molecule into the mesh as a set of triangular faces. A direct mesh construction algorithm is then used to refine the existing mesh in the neighborhood of the dielectric interface. This will allow an accurate calculation of the induced polarization charge to be carried out while maintaining a sparse grid structure in the rest of the computational space. The inclusion of an ionic boundary at some finite distance from the dielectric interface can be automatically achieved as the grid point distribution outside the solute molecule is constructed using a set of surfaces topologically equivalent to this boundary. The meshes obtained by applying the algorithm to real molecular geometries are described. *J Comput Chem* **18**: 1570–1590, 1997

Keywords: dielectric continuum; Poisson–Boltzmann equation; finite element; mesh generation; grid generation

*Present address: Department of Biochemistry and Molecular Biophysics, Columbia University, P&S, New York, NY 10032

Correspondence to: R. A. Friesner

Contract/grant sponsor: National Institutes of Health; contract/grant number: GM-42018

Contract/grant sponsor: FCAR

CCC 0192-8651 / 97 / 131570-21

Introduction

In recent years, there has been a renewed interest in the application of classical electrostatics methods to the study of molecular properties in solution. Classical force fields used to represent both inter- and intramolecular interactions, which include electrostatic interactions, have played a major role in the development of computer simulations of molecular systems. However, their use in the study of solvation phenomena is somewhat limited by the significant computational cost associated with the inclusion of a large number of solvent molecules in a simulation. The Poisson–Boltzmann (PB) equation and the dielectric continuum solvent framework have provided an alternative to explicit solvent descriptions of solvated systems and extensive use has been made of this formalism to the study of molecular properties.[1] By representing a polarizable solvent as a continuum with given dielectric properties, the problem of evaluating the interactions between the solute and solvent are reduced to solving numerically a three-dimensional partial differential equation (PDE) for the electrostatic potential $\phi(\vec{r})$. With the potential given in units of k_BT/e, the specific form of the equation which has been used is:

$$\nabla \cdot (\epsilon(\vec{r})\nabla \cdot \phi(\vec{r})) - \epsilon(\vec{r})\kappa(\vec{r})^2 \sinh[\phi(\vec{r})] = -\frac{4\pi}{k_BT}\rho^f(\vec{r}) \quad (1)$$

where $\epsilon(\vec{r})$ is the spatially varying dielectric function, ρ^f is the solute *fixed* charge distribution, k_B is the Boltzmann constant, T is the temperature, and e is the proton charge. The presence of free ions in the solution (in the case of a 1–1 salt) is accounted for by the sinh term, and $\kappa(\vec{r})^2 = 8\pi e/\epsilon kT$. As many applications are to systems surrounded by weak ionic solvents, the equation can be linearized as follows:

$$\nabla \cdot (\epsilon(\vec{r})\nabla \cdot \phi(\vec{r})) - \epsilon(\vec{r})\kappa(\vec{r})^2 \phi(\vec{r}) = -\frac{4\pi}{k_BT}\rho^f(\vec{r}) \quad (2)$$

Most often, only the case of pure aqueous solvation is of interest so that the problem simply reduces to Poisson's equation.

Because an analytical solution can only be obtained for simple geometries, the utility of the model rests on the ability to obtain a numerical solution to the equation for arbitrary molecular shapes. Several methods have been proposed and fall into two general categories. The first set includes methods which rely on the use of three-dimensional grids and are based on a finite difference discretization of the problem. These were first introduced by Warwicker and Watson.[2] Improvements to the finite difference method for the PB equation were subsequently proposed by Klapper et al.,[3] Gilson et al.,[4] and Nicholls and Honig.[5] More recently, the development of a multilevel method for the problem by Holst and Saied[6] has significantly improved the convergence properties of the finite difference formulation for the nonlinear problem as well as the overall performance of the method.

The second set, known as boundary element methods, are only applicable to the Poisson equation. These rely on the use of two-dimensional surface grids and are based on a reduction of the dimensionality of the problem: the three-dimensional PDE is transformed into a two-dimensional integral equation. The approach was first proposed by Zauhar and Morgan.[7] Although this reduction appears to be an attractive alternative, it leads to a dense set of linear equations which are computationally expensive to solve iteratively.[8] Furthermore, the use of a boundary element formulation is only possible for the complete problem (ionic solvents) when it is coupled to a three-dimensional finite element grid in the regions of space accessible to the ions.[9]

At present, if we consider only three-dimensional grids, there appears to be no definite answer to the question of which of structured or unstructured grid techniques are more efficient in the numerical solution of PDEs in complex geometries. A great deal depends obviously on the specific problem at hand but also on the implementation of the different methods used. However, in spite of the resulting programming complexity, the use of unstructured three-dimensional grids has significantly increased in the last decade. These techniques have proven particularly useful when applied to problems requiring locally very accurate numerical solutions in geometries featuring very different length scales. In such situations, covering the entire computational space at the resolution corresponding to the smallest relevant length scale in the problem would be hopelessly inefficient and sometimes impossible because of various hardware limitations.

The problem we consider here differs somewhat from typical problems involving the numerical solution of different PDEs, because we are concerned with a model of solvation in which integrals of the solution over space and their derivatives with respect to atomic coordinates are the desired quantities, and the fine resolution features of the solution to the partial differential equation are unimportant. Therefore, a structured grid method itself need not be of high resolution to be of great utility and may benefit substantially from the use of recursive multilevel methods which are virtually impossible to systematically implement in an unstructured grid approach.

However, for the Poisson equation, to which the PB equation reduces in the case of pure aqueous solvation, much of the information needed to compute the unknown quantities is concentrated in the neighborhood of the molecular surface. Our aim has been to design an unstructured and highly uneven three-dimensional grid to achieve an almost volume-to-surface-like reduction in complexity of the problem while retaining the desirable feature of reduction to a sparse linear system. The specific algorithm we present constructs a constrained three-dimensional mesh in which the molecular surface and the ion exclusion boundary can be represented as sets of triangular faces belonging to the mesh elements.

The tetrahedral mesh will be used to solve the linear Poisson–Boltzmann equation numerically using a Galerkin finite element formulation. In this approach the solution is expanded in linear basis functions on the elements and the weak form of the PDE is used to construct a sparse set of linear equations for the expansion coefficients. This formulation also enables accurate calculations of the polarization energy gradient to be carried out by using the analytical relationship between the positions of the grid vertices in the computational space and the atomic positions. The general method and applications to *ab initio* solution phase geometry optimizations using self-consistent reaction field methods has been discussed in a separate publication.[10]

In the next section of this article we describe the geometry of the problem and recall the algorithms used for constructing both the Richards molecular surface, which will represent the solute–solvent dielectric interface, and the ion exclusion boundary. We then describe the method used to construct a nonuniform three-dimensional grid adapted to the molecular geometry. The design and implementation of a mesh generator using combined Delaunay triangulation and direct meshing techniques will then presented. Applications of the method are discussed in the fifth section, while results of finite element calculations using these meshes are presented in another article.

Dielectric Continuum Model and Construction of Domain Boundaries

DEFINITIONS

The application of the dielectric continuum solvent model to the study of complex molecules requires a precise definition of the boundary between the molecule and its environment. For a single atom or ion, this boundary may be taken to be a sphere of radius equal to the Van der Waals or ionic radius of the element.[11] Alternately, the atomic radii may be determined by fitting the properties predicted by the model to experimental values or to the result of *ab initio* calculations. By extension, the corresponding boundary for a molecule would then simply be represented by the unobstructed part of the union of the atomic spheres.

However, this simple picture is inadequate to describe the boundary of the region accessible to any part of a solvent molecule. Any model of the molecular surface must follow, in some way, the boundary of the space from which other molecules are excluded. The proximity of two atoms may exclude the presence of a molecule in a given area because of short range repulsive interactions, even though the atoms may not overlap; that is, the relative distance between the atoms may be greater than the sum of the atomic radii. Clearly, the atomic sphere overlap fails to correctly describe this situation.

A more realistic definition for the molecular surface was first proposed by Richards[12] in 1977. The object he constructed was in fact the surface described by a point on the surface of an idealized spherical solvent probe as it is rolled around the solvated molecule. The part of this surface on which the solvent sphere is in contact with a single atom is referred to as the *contact surface*, whereas the part formed by the solvent surface when it is in contact with more than one atom is referred to as the *reentrant surface*. The surface described by the probe center in the process of rolling it around the molecule is called the *solvent-accessible surface*. The relation between the different surfaces is illustrated in Figure 1.

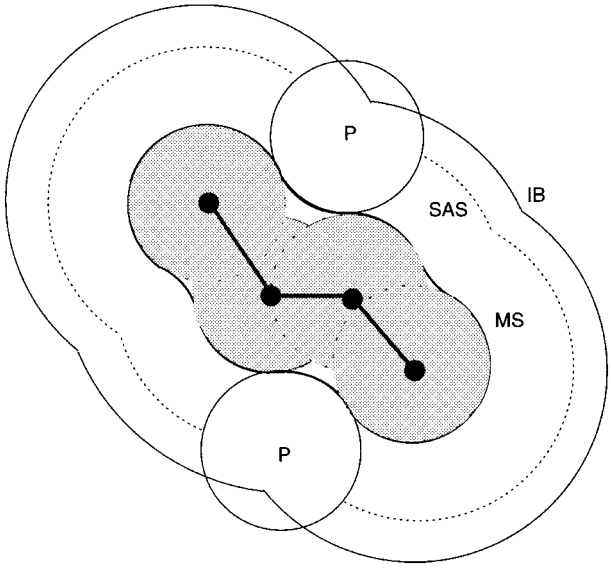

FIGURE 1. Molecular geometry and surfaces. The dotted line indicates the location of the solvent-accessible surface (SAS). The inner and outer solid lines indicate the Richards molecular surface (MS) and ionic boundary (IB). The boundary of the gray area is the Van der Waals surface. Two locations of the probe which generate parts of the reentrant surface are indicated by spheres labeled P.

The number of possible contact points between the probe and the molecule is, in most cases, one, two, or three. A larger number of probe–solute contacts is possible only if there are four or more points belonging to different atomic spheres lying on a sphere of radius exactly equal to the probe radius. Consequently, the surface is a piecewise smooth object constructed from three different types of objects: a section of an atomic surface; a section of the probe surface; and a section of a torus. In the first case, the surface is convex, whereas in the last two cases it is concave. As discussed by Connolly,[13] this distinction can be interpreted in terms of the number of degrees of freedom available to the probe center when it is in contact with the molecule. This number is then three minus the number of contacts formed between the probe and the molecule.

With this definition, the molecular surface is C^0 continuous and can enclose either a simply or multiply connected region of space. The presence of cusps and the occasional multiple connectedness pose considerable difficulties in the process of generating an appropriate three-dimensional mesh for this problem.

TABLE I.
Probe Surface Mobility and Relation to Categories of Surface Elements.

Type	N_{AC}	N_{FP}	Surface element	Shape
I	1	2	Spherical section	Convex
II	3	0	Spherical section	Concave
III	2	1	Torus section	Concave

N_{AC}: number of contacts between the spherical probe and the solute; N_{FP}: number of degrees of freedom of the probe sphere on the surface.

In applications involving ionic solvents, the region of space from which free ions are excluded is bounded by the *ion-accessible surface*. If all ions are assumed to be represented by a point charge inside a sphere of finite radius, the latter surface has the same topology as the solvent-accessible surface; that is, it is the locus of the center of the ionic sphere as it is rolled around the molecule. In the dielectric continuum solvent model, the continuum outside this boundary is assigned the bulk value of the ionic concentration. The relation between the ion exclusion surface and other surfaces is described in Figure 1. Its construction and inclusion in the mesh will be discussed later.

The algorithm described in what follows is used to process the molecular geometry and construct the objects necessary to represent and sample the Richards surface with a finite number of vertices. The surface is uniquely determined once the probe size and the atom center locations and assigned radii are specified. This information can be translated into the location of a finite set of vertices which can be viewed as the basic elements necessary to locate any number of points on the surface. We will refer to these vertices as *control points*. They fall into three classes which correspond to the surface element types discussed previously. The control point types are listed in Table I, after Connolly.[13]

TYPE I SURFACE ELEMENTS: CONVEX SPHERICAL SHELL SECTIONS

The basic building blocks for sampling sections of spherical surfaces in the grid generation process are the Lebedev grids for Gaussian quadrature of spherical harmonics. The order of the spherical harmonic l determines the number of points used

to describe a spherical shell and will be used as a measure of resolution in our surface and grid generation algorithms. The choice of the Lebedev shells was motivated by the need to obtain a controllable near-uniform point distribution at a fixed distance from atomic centers to avoid biasing the mesh in certain angular directions. For a detailed description of the point distribution structure in these grids, the reader is referred to the original work of Lebedev.[14, 15]

We begin by placing vertices on a shell of radius R_i around each atom i located at position $\vec{r}_i$, where $R_i = \sigma_i + r_p$. Here, σ_i is the radius assigned to atom i and r_p is the radius of the probe sphere (Fig. 2 IA). All vertices centered on atom i, located within R_j of atom j, are then discarded. The remaining set of vertices represents a sampling of the solvent-accessible surface of the molecule (Fig. 2 IB). Atoms for which there are no surviving vertices at the maximal resolution available are assigned to the interior of the molecule. A truncated set of atoms, hereafter referred to as surface atoms, is now available to continue the surface construction. The radii of the shell sections centered on surface atoms are subsequently rescaled to the value of the atomic radius (Fig. 2 IC).

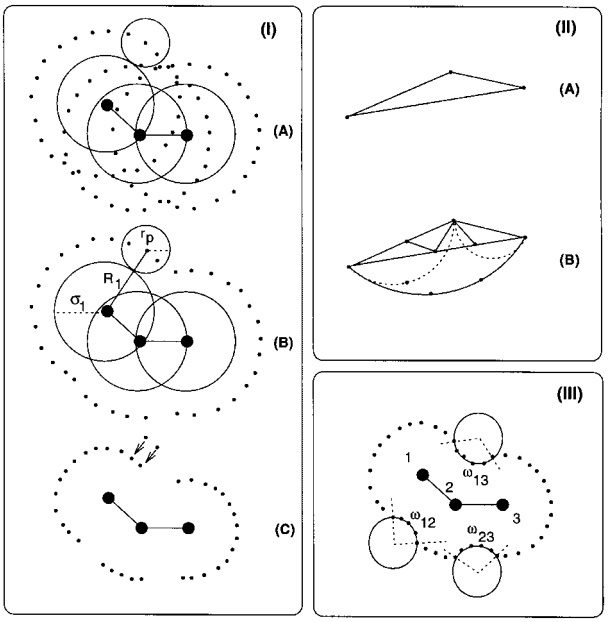

FIGURE 2. Construction of surface elements. Type I elements: (A) shell placement; (B) truncation; (C) rescaling. Sizes of probe and atomic spheres are shown by solid circles. Type II elements: (A) triangle formed by three atom–probe contacts; (B) recursive subdivision produces a pattern which is projected onto the concave spherical shell section. Type III elements: arcs ω_{12}, ω_{13}, and ω_{23} are sampled to generate the concave saddle regions of the Richards surface.

TYPE II SURFACE ELEMENTS: CONCAVE SPHERICAL SHELL SECTIONS

For each surface atom i that has been identified at this point, we construct the set of concave spherical shell sections that are formed by the set of control points of type II for which the probe has a contact with atom i. The method used here is similar to the one described by You and Bashford.[16] Denote r_{ij} the distance between atoms i and j. Consider an atom j_i whose solvent accessible sphere overlaps with that of atom i (i.e., $r_{ij} < R_i + R_j$), and denote C_{ij} the overlap circle. The center O_{ij} and radius $R_{C_{ij}}$ of circle C_{ij} are obtained from:

$$\begin{aligned} d_{O_{ij}} &= \frac{r_{ij}^2 + R_i^2 - R_j^2}{2r_{ij}} \\ \vec{r}_{O_{ij}} &= \vec{r}_i + d_{O_{ij}}\hat{r}_{ij} \\ R_{C_{ij}} &= \sqrt{R_i^2 - d_{O_{ij}}^2} \end{aligned} \tag{3}$$

There will be a subset of the surface atoms k_{ij} which will intersect C_{ij}. Each intersecting atom will occlude an arc a_k^{ij} of C_{ij} (Fig. 3A).

The position of the endpoints of the arc can be obtained by fixing the origin at $\vec{r}_{O_{ij}}$, rotating $\hat{r}_{ij}$ into the $\hat{z}$ axis, and solving the system of equations:

$$\begin{aligned} x^2 + y^2 &= R_{C_{ij}}^2 \\ (x - x_{k_{ij}})^2 + (y - y_{k_{ij}})^2 &= R_k^2 - z_{k_{ij}}^2 \end{aligned} \tag{4}$$

There are three cases to consider:

- If $x_{k_{ij}} = 0$ and $y_{k_{ij}} = 0$, then a solution exists only if $R_{C_{ij}}^2 = R_{k_{ij}}^2 - z_{k_{ij}}^2$ and the overlap is the entire circle. This case is considered equivalent to C_{ij} being completely occluded by atom k.
- If $x_{k_{ij}} = 0$ and $y_{k_{ij}} \neq 0$, then:

$$\begin{aligned} y &= \frac{R_{C_{ij}}^2 - R_{k_{ij}}^2 + y_{k_{ij}}^2 + z_{k_{ij}}^2}{2y_{k_{ij}}} \\ x &= \pm\sqrt{R_{C_{ij}}^2 - y^2} \end{aligned} \tag{5}$$

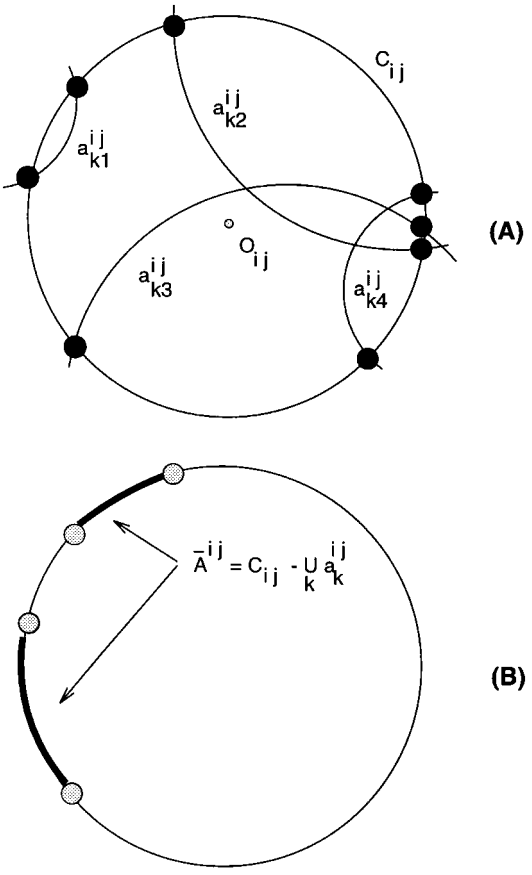

FIGURE 3. Construction of type II control points. (A) Set of arcs occluding overlap circle C_{ij}. Possible type II control points are shown as black circles. (B) Surviving control points shown as gray circles after segment overlap is determined. The set $\bar{A}^{ij}$ is composed of the thick circle arcs.

- If $x_{k_{ij}} \neq 0$, then:

$$y = \frac{1}{1+\gamma^2}\left(\zeta\gamma \pm \sqrt{R^2_{C_{ij}}(\gamma^2+1) - \zeta^2}\right) \quad (6)$$

$$x = \zeta - \gamma y$$

where:

$$\zeta = \frac{R^2_{C_{ij}} - R^2_{k_{ij}} + x^2_{k_{ij}} + y^2_{k_{ij}} + z^2_{k_{ij}}}{2\,x_{k_{ij}}} \quad (7)$$

$$\gamma = \frac{y_{k_{ij}}}{x_{k_{ij}}}$$

We denote A^{ij} the set of these arcs. Now consider the fraction of C_{ij} left exposed by the union of all these arcs, which is itself a set of arcs. We denote it $\bar{A}^{ij}$ and we have:

$$\bar{A}^{ij} = C_{ij} - \bigcup_k a_k^{ij} \quad (8)$$

There are two possibilities:

- If $\bar{A}^{ij} = \varnothing$, or if there exists an atom k which does not intersect C_{ij} but which contains C_{ij} completely, there are no control points generating concave spherical sections of the surface, depending on the pair of atoms (i, j).
- If $\bar{A}^{ij} \neq \varnothing$, then the endpoints of each distinct arc in $\bar{A}^{ij}$ are the control points generated by contacts between the probe and atoms $\{i, j, k_{ij}\}$.

To merge the segments in A^{ij} and obtain the elements of $\bar{A}^{ij}$, we represent them by their angular boundaries on C_{ij}. We have $a_k^{ij} = [\alpha_k^{ij}, \beta_k^{ij}]$ with the segment interior representing the occluded part of C_{ij}. The delimiters are then sorted by angular value. For each ordered segment initiator, α_k^{ij}, the circle is searched counterclockwise until a segment terminator, $\beta_{ij}^{k'}$, is found. All segment boundaries x_l^{ij} such that $x_l^{ij} \in [\alpha_k^{ij}, \beta_{k'}^{ij}]$ are deleted from the list, and replaced with a new merged segment $[\alpha_k^{ij}, \beta_{k'}^{ij}]$. The process is repeated until the remaining segments contain no other segments in the list. The segment merging step is illustrated in Figure 3.

Each control point found in this process will generate a concave spherical section of the surface that must now be sampled. Given a type II control point, the three contact points with atoms $\{i, j, k\}$ form a triangle that can be recursively subdivided by bisecting the longest side until the longest side of every subtriangle is smaller than some resolution parameter. The final list of triangles is then projected onto the surface of the probe sphere. This process is similar to the one implemented by Connolly[13] in his surface triangulation and is illustrated in Figure 2.II.

The sampling resolution must be scaled to reflect the resolution used in regions of type I. Given atoms $\{i, j, k\}$ from which a given reentrant region is generated, the same angular point distribution

will be found on all three atoms. Because the assigned radii are not necessarily identical for all three atoms, the average spacing between nearest vertices on the atom surfaces will be different. A measure of this distance is the fraction:

$$\lambda_i = \sqrt{\frac{4\pi\sigma_i^2}{N_l}} \tag{9}$$

The edge subdivision process is then repeated until there are no edges left on subtriangles exceeding $\frac{1}{3}(\lambda_i + \lambda_j + \lambda_k)$ in length.

TYPE III SURFACE ELEMENTS: CONCAVE SADDLE SECTIONS

The reentrant sections of the molecular surface consisting of sections of tori depend on curves described by the probe center rather than on fixed positions of the probe center. To construct these regions, the exposed arcs of the overlap circle C_{ij} are sampled to generate a finite set of control points. Each type III control point will generate an arc of circle $\omega_{ab} = [\phi_a, \phi_b]$, perpendicular to C_{ij}. The three possible angular boundaries for the arc are:

- If $0 < d_{O_{ij}} < r_{ij}$, then:

$$\phi_a = \frac{\pi}{2} - \cos^{-1}\alpha_j \quad \phi_b = \frac{\pi}{2} + \cos^{-1}\alpha_i \tag{10}$$

- If $0 < r_{ij} < d_{O_{ij}}$, then:

$$\phi_a = \frac{\pi}{2} + \cos^{-1}\alpha_j \quad \phi_b = \frac{\pi}{2} + \cos^{-1}\alpha_i \tag{11}$$

- If $d_{O_{ij}} < 0 < r_{ij}$, then:

$$\phi_a = \frac{\pi}{2} - \cos^{-1}\alpha_j \quad \phi_b = \frac{\pi}{2} - \cos^{-1}\alpha_i \tag{12}$$

where:

$$\alpha_i = \frac{R_{O_{ij}}}{R_j} \quad \alpha_j = \frac{R_{O_{ij}}}{R_i} \tag{13}$$

As for type II surface elements, the sampling resolution is adjusted to match the angular resolution on atoms $\{i, j\}$. The linear spacing on the arcs of circle is taken to be $\frac{1}{2}(\lambda_i + \lambda_j)$.

Construction of Molecular Grid

EMBEDDED SHELL STRATEGY

The surface construction algorithm just described can now be extended and used to build a three-dimensional adaptive grid for our problem. One can think of such a grid as composed of shells (the term is used in a loose sense here as the objects can in some cases be multiply connected) of different ''radii.'' A three-dimensional grid, for a molecule consisting of a single atom, can be easily assembled by placing concentric shells of Lebedev vertices around the atomic center up to some finite distance. Similarly, for a nontrivial molecule, any assigned set of radii $\{\sigma_i\}$ will generate a unique surface $S(\sigma_i)$, which can be chosen to be topologically equivalent either to a Richards or to an accessible surface. By assigning the same number, N, of radial shell positions $\{\sigma_i^l,\ l = 1, N\}$ to each atom, i, we can construct a set of surfaces, $\{S_l(\sigma_i^l)\}$, enclosing each other. In other words, if $\sigma_i^n < \sigma_i^m$ for $n < m$, then S_m will enclose S_n. By choosing one of the σ_i^l to be the physical radius $\sigma_{s,i}$ of atom i, a three-dimensional grid is constructed which contains a sampled representation of the internal boundary in the problem.

The use of the Richards surface model is unsuitable as a grid subset for all shell positions for two reasons. First, the model is meant to represent the molecule/solvent interface, so for shell positions lying within the physical radius of an atom there is no atom/probe contact possible. Second, for small values of σ_i^l, more precisely for $\sigma_i^l \sim \lambda_i$, it becomes impossible to represent the corresponding surface with a near-homogeneous point distribution with a discretized representation. The atomic shells in this case are polyhedra with a small number of faces and one can no longer think of them as spherical point distributions. The interior of the molecule (surfaces generated from positions $\sigma_i < \sigma_{s,i}$) will therefore be represented simply by a union of atomic grids (no reentrant parts). In this region, the grids are truncated at the Voronoi surfaces of the atoms. For simplicity, we choose the surfaces used to construct the grid in the region exterior to the molecule to be topologically equivalent to the solvent-accessible surface. In numerical applications, we have found that the precise form of the exterior grid point distribution has very little effect on the electrostatic energies calculated from the numerical solution to the PDE. Further-

more, for the case of ionic solvents with associated ionic radius r_{ion}, the ion exclusion boundary is automatically constructed by including the surface $S(\sigma_i^n + r_{ion})$ in the generation of the exterior grid.

To obtain a triangulated representation of the Richards surface in the mesh generation process, we found it necessary to insert images of the surface in the grid. These will be required during the Delaunay triangulation stage and will serve to generate tetrahedral elements composed of three surface vertices out of four (see discussion of unstructured mesh generation method). The corresponding faces form the triangulated representation of the Richards surface. These shells are constructed differently from the ones described previously. The normal vector to the surface at each point is used to construct an image of each surface point at some fixed distance from it. In most cases, one interior image is sufficient to allow the generation of the constrained mesh. In more complicated geometries a second exterior one is also necessary. Finally, it should be noted that these surface images are distinct from the ones that are created during the mesh refinement stage which will be discussed in what follows.

GRID GENERATION PARAMETERS

We now turn to the problem of automatically assigning the shell positions and corresponding Lebedev resolution parameters to the various sections of the grid. The flexibility of the resolution assignment is severely restricted by the mesh generation stage of the problem so that complete automation may eventually only be possible if the mesh generation results are used as feedback to modify the chosen grid resolution parameters. In the current implementation, some guidelines must be followed in the resolution assignment. These generally lead to usable meshes but there is no way of determining *a priori* whether or not the mesh construction will fail for a given point distribution.

We define a minimal set of grid parameters with an associated tolerance range in Table II. The data set representing the molecule and the solvent probe sphere and these parameters completely specify the structure of the grid obtained using the algorithm we have described to date.

Parameters g_{d0}, g_{d1}, and g_{d2} are specified in units of the resolution parameter λ [eq. (9)] for a prototype atom radius, $\sigma_{s,i0}$, usually the carbon radius 1.9 Å. With these definitions it is straightforward to distribute the shell positions as follows:

- *Interior*. Distribute $n_1 = n_{sf} - 1$ shells evenly on atom i between positions g_{d0} and $\sigma_{s,i} - g_{d1}\lambda_{s,i0}$.
- *Molecular surface*. Place a single shell at position $\sigma_{s,i}$ on atom i.
- *Exterior*. Distribute $n_2 = n_{sh} - n_{sf}$ shells evenly on atom i between positions $\sigma_{s,i} + g_{d2}\lambda_{s,i0}$ and $\sigma_{s,i} + g_{db}$. If an ionic boundary is required, shift positions of exterior shells so that second shell is placed at $\sigma_{s,i} + r_{ion}$.

The surface images described in the section on enclosing shells are not counted in the number of shells n_{sh}. A zero value for parameter g_{d1} or g_{d2} indicates that the corresponding image is omitted.

TABLE II.
Grid Generation Parameter Definitions and Ranges Allowed.[a]

Parameter	Units	Range	Description
n_{sh}	—	4–15	Total number of shells in grid
n_{sf}	—	2–7	Molecular surface shell position
l_{sf}	—	8–29	Lebedev grid index at molecular surface
g_{d0}	Å	0.01–0.5	Atom center, first shell spacing
g_{d1}	$\lambda_{s,i0}$	0.3–0.5	Interior image, molecular surface shell spacing
g_{d2}	$\lambda_{s,i0}$	0.3–0.7	Molecular surface shell exterior image spacing
g_{db}	Å	4.0–15.0	Molecular surface, boundary spacing
α_1	—	0.0–2.0	Interior point density scaling parameter
α_2	—	−2.0–0.0	Exterior point density scaling parameter

[a]This set completely specifies the grid point distribution to be triangulated.

The limited flexibility in the grid construction lies in the possibility of varying the Lebedev angular resolution parameters with the shell position. Instead of specifying the n_{sh} different Lebedev parameters associated with the shells centered on each atom, we automate the assignment with distribution functions. Given the number of vertices assigned to a spherical shell on the molecular surface N_s, the target number of vertices on a spherical shell of radius r is given by:

$$N_t = N_s \Theta\left(\left(\frac{r}{\sigma_{s,i0}}\right)^{\alpha}\right) \tag{14}$$

where $\Theta(x) = n,\ n \in N$ if $x \in [n - \frac{1}{2},\ n + \frac{1}{2})$. The parameter α is either α_1 or α_2 as specified in the Table II for computing the interior or exterior angular point distributions. The final choice of the Lebedev grid is made by selecting the spherical harmonic index l for which the number of vertices N_l is closest to N_t. The same number of vertices N_s is assigned to the surface shell of each atom. The range of values of N_s is given by $6 \le N_s \le 302$, and is determined by the Lebedev grid index used for the surface shells l_{sf}, $1 \le l_{sf} \le 29$.

MOLECULAR SURFACE DISCONTINUITIES

As indicated by Connolly,[17] the intersection of surface regions of type II can generate cusps in the molecular surface which require special treatment. These pose a serious problem in the constrained triangulation process because of the finite resolution used to describe the object. The reason is that the ability to generate a constrained mesh rests on an unambiguous identification of the spatial regions. In other words, elements must be easily classified into interior or exterior, with some constructions being forbidden. The rapid variation of the molecular surface in cusp regions can cause unpredictable breakdowns of the algorithm because of the formation of ambiguous structures. At present, we have resorted to cusp trimming[17] to handle these situations. The procedure consists essentially in screening type II surface elements and discarding all vertices belonging to the overlap between these elements. We have found this screening to reliably allow the construction a triangulated molecular surface in the mesh generation process when the algorithm is applied to molecules of up to 200 atoms. However, for more complex structures such as proteins, the presence of cusp regions with complicated topologies occasionally prevents an unambiguous identification of the Richards surface in terms of tetrahedral element faces. Work is presently underway to circumvent this difficulty.

INTERNAL CAVITIES

In applications involving proteins, the Richards surface definition can sometimes lead to the construction of internal cavities when gaps comparable in size to the solvent probe are present in the interior of the molecule. If these cavities are sufficiently large, they will be described by mesh elements assigned solvent dielectric properties. There is, at the moment, no special provision in the mesh generation approach to reassign these to the molecular interior. However, it is always possible, once these are detected by other means, to explicitly place solvent molecules within them (as internal waters can often be found in the crystal structures of proteins in such situations) and to apply the present strategy to the resulting supramolecule. Alternately, improper assignment could also be avoided by using expanded radii for the interior atoms surrounding the cavity. If the cavity lies deep inside the protein, the local use of expanded radii will have no effect on the construction of the protein–solvent dielectric interface.

Automated Mesh Generation

CHOICE OF STRATEGY

In the context of finite element calculations, it has been established that tetrahedral elements alone provide enough flexibility to construct meshes for arbitrary geometries. Although the use of cubic, prism-like, or curved elements is possible in principle, these are better suited to applications involving simple geometries or to cases where a convenient analytical representation of the boundaries in the problem is available.

In the process of constructing a PB solver for an arbitrary molecular geometry we must allow for the fact that the only common grid building unit between different systems is the individual atom. In this sense the same algorithm must be able to construct a suitable mesh for all cases ranging from a single atom to a protein. The variability in molecular shapes prevents us from constructing anything but an atom-based description of the regions involved in the calculation. One conse-

quence of this requirement is that we must exclude any numerical approach to the grid generation,[18] because there is no convenient way of constructing *a priori* a curvilinear coordinate system following the molecular geometry. This process would require solving a Poisson-like problem for each curvilinear coordinate, which would compensate, in computational cost, any gain made from the use of an unstructured grid approach.

We begin by briefly reviewing some of the methods for three-dimensional mesh generation. Although a variety of algorithms are available for the generation of two-dimensional triangulations, many of them based on coordinate transformations or on the use of other tools,[19,20] the choice of methods to generate three-dimensional meshes is more restricted because of the increased complexity of the problem. One can distinguish three general categories of algorithms of which several two- and three-dimensional implementations can be found.

ADVANCING-FRONT METHODS

These methods are based on the iterative modification of an existing triangulated boundary. In n dimensions, the process begins with a triangulated representation of an object of dimension $n-1$ referred to as a front. The n-dimensional mesh is initialized with the formation of an initial element by connecting a vertex in the grid to an $n-1$ simplex of the front. By adding the newly formed faces to the front and deleting all faces connected to vertices surrounded by n-simplices, a new front is generated. The process terminates when there are no more grid vertices available to link to. The details of the method and its implementation are described by Löhner et al.[21,22] Our surface construction algorithm produces a sampled representation of the molecular surface, but does not provide us with a triangulated surface. In fact, the only practical way of constructing a triangulated molecular surface without extracting it from a three-dimensional mesh is through a marching cubes algorithm.[23] Because the resulting surface vertices would not lie on the surface itself but on the nearest grid lines at a given resolution, the triangulated pattern produced by the marching cubes algorithm would have to be projected onto our sampling of the surface. Because of the numerous problems that would arise from the possible local mismatch in resolution, we have chosen not to follow this approach.

HIERARCHICAL AND FINITE-OCTREE METHODS

The process of hierarchical subdivision of complex geometries into simpler ones is one of the fundamental approaches to computation involving complex geometries. An adaptively refined octree structure is a natural framework for organizing the spatial information needed to described complicated shapes. It would seem equally natural then to base a mesh generation strategy on this idea.

HIERARCHICAL AND FINITE-OCTREE METHODS

The process of hierarchical subdivision of complex geometries into simpler ones is one of the fundamental approaches to computation involving complex geometries. An adaptively refined octree structure is a natural framework for organizing the spatial information needed to described complicated shapes. It would seem equally natural then to base a mesh generation strategy on this idea.

One of the most complete descriptions of the method has been provided by Shephard and Georges.[24] Their algorithm can be divided into two stages. First, the computational space is hierarchically subdivided with an adaptive finite octree in which the cell subdivision is guided by the geometry of the reference object. Second, the terminal octants are meshed internally and are assigned element structures compatible with the neighbor octants. The terminal cell size is used to control the mesh resolution.

The only requirement imposed on the octant description of the object is topological equivalence with the original model. In other words, the boundaries of the object must be topologically equivalent to a subset of the octant faces. In this approach, the vertices of the mesh are simply the set of terminal octant corners and the intersections of the object boundaries with the octant edges. This implies that the cell subdivision criterion is determined by how well the element faces resulting from meshing the contents of a terminal octant reflect the object features or boundaries lying within the octant.

Although this approach is sufficiently flexible to construct a suitable mesh for any geometry, a

feature which may be essential in extending our method to more complex proteins with multiply connected surfaces, it does not permit *a priori* specification of all the mesh point locations. Because this is a necessary step in our adaptive grid strategy we cannot follow the adaptive octree approach in the triangulation process. However, as we will see the concept of adaptive octree will remain an invaluable tool in developing an efficient constrained triangulation algorithm for our problem.

AUTOMATIC DELAUNAY TRIANGULATION AND BOWYER'S ALGORITHM

We now turn to the third class of existing mesh generation strategies. Generally, these algorithms are based on the construction of a triangulation derived from the Voronoi diagram construction[25] of the predetermined grid point distribution. The relationship between the Voronoi diagram and the associated Delaunay triangulation (DT) are represented schematically in Figure 4 and are defined as follows.

Consider an arbitrary distribution of n vertices in N-dimensional space V_n. For every pair $(v_i, v_j;\ i \neq j)$ of vertices, the plane P_{ij} perpendicular to $\vec{r}_{ij}$ through the midpoint of r_{ij} divides all points in space into three sets according to their distance d to vertices v_i and v_j. For any point p in space, either $d_{pi} < d_{pj}$, $d_{pi} > d_{pj}$, or $d_{pi} = d_{pj}$, in which case $p \in P_{ij}$. The set of points satisfying $d_{pi} < d_{pj}$ $(j = 1, \ldots nj \neq i)$ is the subset of the N-space closer to vertex i than to any other vertex and is referred to as the *Voronoi cell* of point i.

Now, consider a set of $N + 1$ vertices $C_\alpha = \{v_j,\ j = j_1, \ldots j_{N+1},\ v_j \in V_n\}$ such that the intersection between the N hyperplanes $\{P_{j_k j_l}\, j_k \neq j_l\}$ exists. This point, which we label O_α, is equidistant to all vertices in C_α and is therefore the center of a hypersphere S_α passing through these vertices. This hypersphere will be referred to as the *circumsphere* of the *N-simplex* C_α. If set C_α is chosen so that S_α encloses no other vertices in V_n, then the mesh formed by $\bigcup_\alpha C_\alpha$ is the Delaunay triangulation of the N-dimensional space.

Whereas several very efficient algorithms have been proposed to directly construct Voronoi diagrams and their associated Delaunay triangulations,[26–29] in three dimensions these are generally only applicable when there are no constraints to be imposed on the triangulation or when the constrained mesh generation problem is two-dimensional.[30] In the latter case, the complexity of the existing algorithms make them unattractive for generalizations to higher dimensions. Instead, the method originally developed by Bowyer,[31] for obtaining the Delaunay triangulation of a set of vertices, can be adapted to generate a constrained triangulation of which only a subset satisfies the Delaunay property. The algorithm is based on two theorems, which are obtained by exploiting the relationship between the Voronoi construction and the associated triangulation. We will only cite them here and the reader is referred to the work of Baker[32] for a rigorous proof.

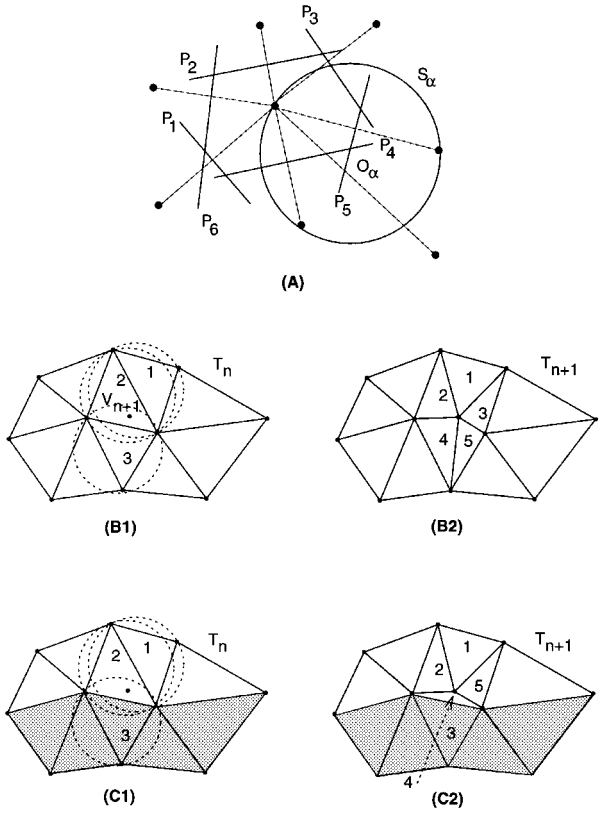

FIGURE 4. Voronoi diagram and Bowyer's algorithm. (A) Voronoi cell of central vertex enclosed by P1–P6. The intersection, O_α, of planes P4 and P5 is the center of the circumsphere of a mesh element. (B1) In Bowyer's algorithm, insertion of V_{n+1} requires removal of elements 1–3. (B2) Reconnection to the cavity faces produces new elements 1–5. (C1) Constrained triangulation: same as (B1), but element 3 is frozen in place. (C2) Element 3 is kept as part of the mesh and new elements are 1, 2, 4, 5.

Consider the Delaunay triangulation T_n of the set V_n. If a vertex v_{n+1} is added to this set such that v_{n+1} lies inside the convex hull of the set V_n,

it will then lie inside one of the simplices C_α. Let O_α be the center of the circumsphere S_α and R_α be its radius. We must then have $d(v_{n+1}, O_\alpha) < R_\alpha$. Let B be the set of circumspheres which contain v_{n+1}; that is, $B = \{S_\alpha | d(v_{n+1}, O_\alpha) < R_\alpha\}$. The set of elements in B form a cavity that will be referred to hereafter as the *Delaunay cavity*. Also, an element face F in the mesh is said to be visible from v_{n+1} if the line segment from v_{n+1} to the center of F traverses no other face in the mesh. With these definitions, the triangulation T_n satisfies the following properties:

1. All faces of the Delaunay cavity B are visible from v_{n+1} so that the region of space covered by B is simply connected.
2. The mesh T_{n+1} obtained by removing B from T_n and connecting v_{n+1} to all the vertices of B is a Delaunay triangulation of V_{n+1}.

These two properties allow the incremental construction of the Delaunay triangulation. By enclosing the entire grid point distribution into a single tetrahedron, the insertion of the first vertex creates a mesh of four elements. By inserting all the remaining points one at a time, identifying the corresponding Delaunay cavity, and updating the mesh according to property 2, the complete mesh can be readily constructed. Furthermore, Baker[32] has shown that, in two dimensions, it is possible to construct a valid triangulation, which does not necessarily satisfy the Delaunay property even if a subset of the elements is fixed in a given configuration. The theorem does not appear to generalize to higher dimensions but it nevertheless appears possible to extend the algorithm itself. Baker reports having successfully applied the idea to the generation of three-dimensional meshes around complete aircraft. As we will see, our implementation makes it equally possible to create constrained three-dimensional meshes for complex molecules.

UNSTRUCTURED MESH GENERATION USING A COMBINED DELAUNAY TRIANGULATION–DIRECT MESHING ALGORITHM

The aim of our implementation of Bowyer's algorithm will be to obtain a subdivision of the computational space in contiguous and nonoverlapping tetrahedra which can be unambiguously assigned to one of the physical regions associated with the continuum electrostatic model. We will use a hybrid approach using some elements of Baker's implementation[32] and of direct meshing techniques. The latter term refers to the process of constructing the elements following a predetermined pattern rather than using a searching technique to build the connections. Our algorithm can be divided into four parts, which we will identify as *initialization*, *tessellation*, *classification*, *and refinement*. All strategies described have been optimized to complete each specific task in the smallest possible number of operations.

Initialization

The driving principle in the implementation of the DT algorithm is to attempt to minimize the number of tetrahedra that must be searched or removed at each point insertion step. This can be accomplished if at any given step, the subset of the final grid which generates the intermediate mesh is almost uniformly distributed in the computational space. Therefore, we require that a sequence of points never be clustered in a given area of the grid during the triangulation stage. If we suppose that there is no requirement on the global insertion order imposed by the need to generate an embedded surface in the mesh, the choice of the point insertion sequence is well defined.

If we label the N vertices by insertion order, then the nth intermediate mesh is formed by the set of vertices p_i, $i = 1, n$. The next vertex selected for insertion, p_{n+1}, must be chosen so that:

$$\min_j \left(d(p_{n+1}, t_j^c) \right) = \max_{l>n} \left(\min_j \left(d(p_l, t_j^c) \right) \right) \quad (14)$$

where t_j^c is the center of the circumscribed sphere to the jth element in the mesh and $d()$ is the distance function. It is immediately apparent that the insertion sequence cannot be generated from this criterion because it would require searching through all possible insertion sequences to find the optimal one. To minimize the initialization time we must use an empirical selection rule which will attempt to disperse the point selection across the grid. We have found that an acceptable scheme is to first make a selection at random from the remaining vertices. If p_{n+1} belongs to a subset of the grid generated from atom a_i, then p_{n+1} is accepted, provided p_n, p_{n-1} were selected from sub-

sets generated from atoms a_j, a_k with $j, k \neq i$, if this is possible, given the list of vertices available for insertion.

For larger structures the selection can be made more stringent by excluding consecutive vertices generated from atoms which are Voronoi neighbors of each other. In the final steps of the mesh generation, the criterion often cannot be satisfied so it must be reduced to excluding vertices based on testing p_n only or simply accepting the random selection.

Embedding the molecular surface into the mesh requires completing the insertion of certain subsets of the grid before others. The case where a single internal boundary (the molecular surface) is present will be described. The algorithm can be extended to include additional boundaries (the ion exclusion boundary) when they are present. We divide the grid point distribution into six subsets as defined in Table III.

These are inserted in the mesh in the order 1, 5, 2, 3, 4; that is, in assigning the insertion sequence, the list of vertices available in each subset is exhausted before moving on to the next subset. The specific choice for the subset sequence will be explained in what follows.

The last initialization step consists in constructing a tetrahedron that encloses the entire set of grid vertices. Bowyer's iterative mesh construction requires that, at each step, the inserted vertex be enclosed by at least one of the element circumspheres, so the outer tetrahedron is necessary to initiate the iteration sequence. These four vertices are not used in the finite element computation.

TABLE III.
Definition of Grid Regions Used in Triangulation Algorithm.

Region	Description
1. Atomic centers	Location of molecular partial charges
2. Molecular interior	Region with vacuum dielectric properties
3. Molecular surface	Connolly surface used as dielectric interface
4. Molecular exterior	Region with bulk solvent dielectric properties
5. Boundary	Surface where boundary conditions are applied
6. Outer tetrahedron	Tetrahedral element enclosing all grid points

Tessellation

All grid vertices are now inserted according to the sequence determined in the initialization stage. By inserting the atoms first, we ensure a coarse-level subdivision of the computational space which will reflect the molecular geometry.

During the construction of the Delaunay cavity, the presence of elements having less than three neighbors (elements for which one or more sides face the exterior of the computational space) requires a larger number of logic operations than if the deleted elements are known to be completely enclosed in the mesh. Once all the boundary points have been inserted, the latter property is satisfied so all vertices in this grid subset are inserted immediately after the atom centers. By inserting all interior and surface vertices before any exterior ones, it is possible to obtain an intermediate mesh in which the molecular surface is represented by a set of triangular faces. The configuration of the mesh in the molecular interior can now be frozen in place and a tiling of the surface can be obtained.

The remaining vertices, which now belong exclusively to the exterior subset, are inserted with a modified remeshing step, which preserves any structure located at the surface or inside it. We have found the efficiency of the tessellation algorithm to be strongly dependent on the implementation details and on the choice of data structures. The purpose of tuning each stage in the algorithm was to achieve linear scaling in the number of vertices to be processed. Our current implementation is described in what follows.

The procedure is based on the central property of the Delaunay triangulation: the circumsphere of each element contains the vertices of the element and no other points in the grid. At each point insertion, a search is made for all the elements with associated circumspheres which contain the new vertex inserted. These elements are then removed from the connection table, leaving a cavity in the mesh, and new elements are added in their place. These are the tetrahedra formed by connecting the new point to the faces of the cavity. Connections are then updated for the new elements and the elements outside the cavity but contiguous to it. The list of elements in the mesh is stored as a doubly linked list to avoid the need for reordering in the removal and updating process. The search for an initial element to be deleted is accelerated by using a finite octree.

In pseudo-code, the essential tasks that must be completed at each iteration of the triangulation

process can be described as:

- `triangulation driver:`

```
for each newp {/* loop over points in grid subset insertion sequence */
  celln ← octinsert(newp)   /* place point in tree, get enclosing cell */
  elmnt ← gettrap(celln)    /* return first broken element */
  elist ← makecave(elmnt)   /* build list of broken elements */
  faces ← cavewalls(elist)  /* build list of cavity faces */
  delcave(elist)            /* delete cavity from element list */
  reconnect(faces)          /* connect new point to faces */
}
```

- `octinsert(newp)`: A point is inserted in the tree structure by bisection over its coordinates until the lowest level (smallest available spatial cell) is reached. If the terminal cell already contains n_{max} vertices it is subdivided and its children inherit the fraction of the contents they enclosed. The tree is descended once again. If the final cell contains only the new vertex, then the parent cell is passed on, otherwise the enclosing cell is retained:

```
celln ← top_of_tree
celln ← descend_tree(newp,celln)
if (celln.contents==FULL) {
  subdivide_cell(current_cell)
  current_cell ←
descend_tree(current_cell)
}
if (celln.contents==ONE_VERTEX) {
  return celln.parent
} else {
  return celln
}
```

- `gettrap(celln)`: The list of vertices contained in `celln` and its neighbor cells is assembled [`ocgather()`] and searched to locate `nearp`, the vertex closest to `newp`. Because each point that is already part of the mesh carries a pointer to one element of which it is a vertex, the element associated with `nearp` is used to begin a recursive search [`caveseed()`] for any tetrahedron which is broken by the insertion of `newp`. If no such element is found, the process is repeated by using the parent cell as a starting point. The process terminates when either the first broken element is found or the top of the tree is reached:

```
do {
  nblst ← ocgather(celln)       /* get list of vertices in all cells in area */
  nearp ← neighbor(nblst)       /* find vertex in mesh closest to new point */
  elmnt ← caveseed(nearp)       /* try to find first broken element */
  celln ← celln.parent          /* go one step back up the tree */
} while(elmnt = NOT_FOUND and   /* if not found, expand search list */
        celln != TREE_TOP)      /* until you get to top of tree (all space) */
```

- `makecave(elmnt)`: The contiguity table is used to perform a recursive quadtree search (each tetrahedron has four neighbors) through the mesh for all broken elements, starting from `elmnt`. As broken elements are found they are added to `elist` unless they have been frozen in place to retain the molecular surface shape. The recursive quadtree search is possible because the cavity generated by the point insertion is always simply connected. A nearest

neighbor search will therefore generate the correct list of broken elements:

```
if (elmnt.status = NOT_FROZEN or
     elmnt.status = CHECKED) {
  if (delaunay_test(elemnt) = TRUE) {
    add_element_to_list(elmnt)
    makecave(elmnt.0.1)
    makecave(elmnt.0.2)
    makecave(elmnt.0.3)
    makecave(elmnt.0.3)
  } else return
} else return
```

- `cavewalls(faces)`: The list of elements contiguous to all tetrahedra in the Delaunay cavity that are not in the cavity are assembled in a separate list (`shell`). The faces common to the elements of `elist` and `shell` therefore form the walls of the cavity. As long as the insertion of boundary vertices is not complete some of the cavity tetrahedra may have less than four contiguous neighbors in the mesh. One or more of the cavity faces may then only belong to a tetrahedron in `elist` and not in `shell`. These are identified as the triplets of vertices belonging to the outer tetrahedron OT and are labeled exterior faces:

```
loop over elements in elist (el) {
  if (boundary = NOT_COMPLETED and
       number.of.neighbors(el) < 4) {
    find_exterior_face()
    add_face(faces)
  }
  loop over elements in shell (sh) {
    if (common_face(sh,el))
    add_face(faces)
  }
}
```

- `delcave(elist)`: The set of elements is stored as a doubly linked list terminated by elements that remain unchanged throughout the tessellation. Elements are deleted from the list by bypassing them in the links:

```
loop over elements in elist (el) {
  el1         ← el.forward
  el0         ← el.back
  el0.forward ← el1
  el1.back    ← elo
}
```

- `reconnect(faces)`: A new set of elements is now formed by connecting `newpt` to the Delaunay cavity faces. The contiguity table must be updated so that it may be used in the next step of the mesh generation. This particular task is the bottleneck of the entire triangulation process.

All new elements generated will have `newpt` as a common vertex so that the contiguity information is completely determined by the contiguity of the cavity faces. By keeping track of the number of updated neighbors (`nupde`) for each new element, double checking is avoided and the process is completed in a minimal number of operations:

```
loop over faces (i = 1,nfaces)(fsi){
 if(fsi.nupde < 3){
  loopoverfaces(j=i+1,nfaces)(fsj){
   if (not_yet_linked(fsi,fsj)){
    if (edge_in_common(fsi,fsj)){
     link(fsi,fsj)
     fsi.nupde = fsi.nupde + 1
     fsj.nupde = fsj.nupde + 1
    }
   }
  }
 }
}
```

As long as the cavity removed at each point insertion step is a proper Delaunay cavity, the algorithm will always produce a valid tetrahedral mesh as long as no two vertices in the point set used are identical. A complication arises because of the necessity to freeze the mesh in place once the insertion of the interior and surface vertices is complete. From that point on, the elements lying below the surface are fixed and kept even if they are broken by the insertion of an exterior point lying in the vicinity of the surface. The cavity produced at each step from then on may not necessarily be a proper Delaunay cavity. The mesh generation process can still continue provided there is volume conservation at each step. If the sets of deleted and reconnected elements denoted by $elist[i]$, $i = 1$, $nelist$ and $(newp, faces[i])$, $i = 1$, $nfaces$, then the process must satisfy:

$$\sum_{i=1}^{nelist} vol(elist[i]) = \sum_{i=1}^{nfaces} vol((newp, faces[i])) \quad (15)$$

Once the surface point insertion is complete, this test is performed at every subsequent point insertion. If the test fails, then the point insertion is

canceled and the mesh is left unchanged. The defective vertex is moved at the end of the point insertion list and the algorithm proceeds. In some cases, a small set of vertices is left for which volume conservation has failed for every possible insertion sequence in this set. These points are then permanently discarded from the mesh. A two-dimensional representation of volume conservation failure is shown in Figure 5.

Classification

The part of the mesh that will be used in the finite element calculation is now extracted from the structure we have generated by removing all elements involving connections between the computational boundary and the outer tetrahedron. These elements span a region of space lying beyond the surface on which the boundary conditions are imposed so they are irrelevant to the rest of the calculation.

It is also necessary at this point to identify the interior and exterior regions as they are represented by the elements. This will allow us to extract a triangulated representation of the molecular surface from the mesh. This information will be necessary to refine the mesh along the normal to the molecular surface. Elements are simply labeled interior if they have at least one interior vertex, and exterior otherwise. Because of the constrained triangulation, there are no elements having both interior and exterior vertices. Surface triangles are collected from the set of faces of interior elements which consist of three surface vertices. Because this includes triplets of surface points which may be perpendicular to the plane tangent to the molecular surface at one of the vertices, all triangles that appear twice in this search must be discarded.

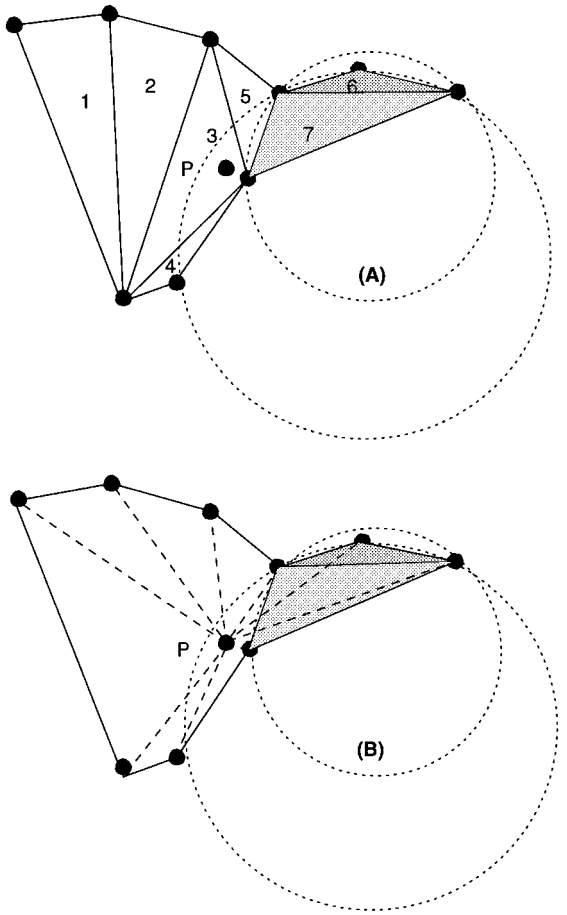

FIGURE 5. Volume conservation breakdown in constrained triangulation. (A) Insertion of vertex P generates the cavity formed by elements 1–6. Element 7 is frozen so the existence of element 6 is possible, making the cavity multiply connected. (B) The reconnection stage generates a set of elements which overlap 7 resulting in an area larger than that of 1–6 (B).

Refinement

The calculation of the induced surface charge, which is necessary in the final evaluation of the polarization energy of the system, requires knowledge of the electric field component, which is normal to the molecular surface. This quantity must be known fairly accurately for the method to be of any practical use. We have found the representation of the electrostatic potential in terms of linear basis functions on the elements to be inadequate for this purpose. Instead, by inserting in the grid four vertices for each surface vertex along the normal to the surface, one can obtain a local second-order finite difference representation of the surface charge. These points could, in principle, be inserted into the mesh at the triangulation stage, but this approach poses two problems. First, because the molecular surface represents 10–30% of the vertices in the grid, this could potentially more than double the computational cost of the triangulation stage. Second, the vertices are placed near the surface and are closely spaced along the normal direction (a spacing of ~ 0.02 Å is typical) so their presence would render the constrained triangulation scheme unworkable. A large fraction of exterior vertices inserted near the surface would violate volume conservation and the mesh genera-

tion process would either break down or be dramatically slowed by the frequent insertion sequence reshufflings.

A more efficient strategy for altering the mesh structure near the surface can be devised by using the surface tiling to generate stacks of triangles which are then linked together in shells of ordered tetrahedra.

Each molecular surface point and associated normal vector is used to generate four vertices, two interior and two exterior. The molecular surface triangulation obtained from the three-dimensional mesh is therefore a valid triangulation for each of the four shells generated in this process, because each surface triangle (v_a^0, v_b^0, v_c^0) has four images (v_a^n, v_b^n, v_c^n), $n = 1, 4$. The exterior images are shown in Figure 6. Triangles belonging to nearest neighbor shells are said to be associated. As shown in Figure 7A, a simple connection pattern will generate three tetrahedral elements between two associated triangles.

Because this pattern is not symmetric with respect to permutation of the vertex labels, it must be compatible with the local tilings generated between all the neighbor triangles. Consider two pairs of triangles belonging to the surface and the

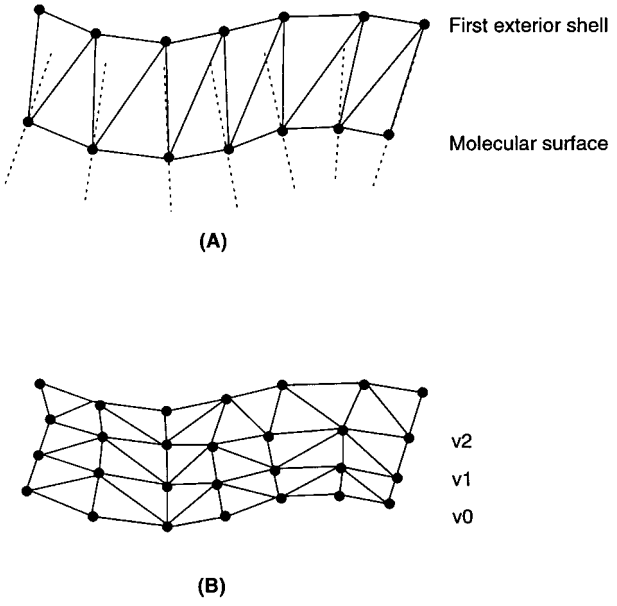

FIGURE 6. (A) Schematic representation of the mesh produced by the constrained Delaunay triangulation. The dotted lines indicate the direction normal to the molecular surface. (B) Local structure of the exterior mesh after refinement. Vertices v0, v1, and v2 are aligned with normal at v0.

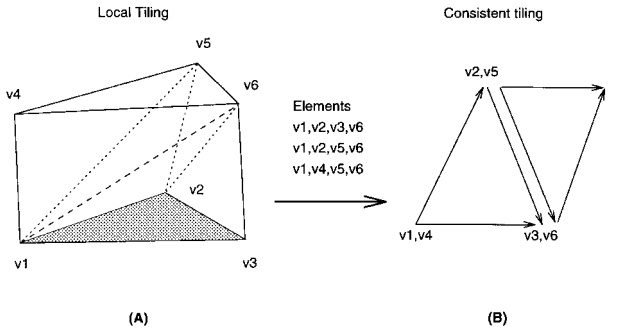

FIGURE 7. Representation of the consistent tiling process. (A) The local tiling generates three tetrahedra for every pair of triangles. (B) The pattern selected for the links must be consistent with the pattern used to generate the tiling from neighboring triangle pairs. This is equivalent to tiling a two-dimensional surface with oriented triangles. Solid arrows represent links across a prism face from the bottom to the top face. Arrow orientation along common edges must match.

first exterior shell, (t_A^0, t_A^1) and (t_B^0, t_B^1), such that each pair forms a prism-like element between the two shells. Suppose also these two prisms, p^A and p^B, share a common face defined by the vertices $(v_a^0, v_b^0, v_a^1, v_b^1)$. The two local tilings created within p^A and p^B must divide this polygon in two. The tilings will be consistent if the division is the same for both prisms so that if (v_a^0, v_b^0, v_a^1) is an element face inside p^A it must also be an element face inside p^B. If the links across shells are represented by arrows, the problem reduces to tiling a closed two-dimensional surface with oriented triangles (Fig. 7).

The algorithm used to complete this task uses the molecular surface connection table. This is the contiguity table for surface triangles. Using this information, a compatible set of local tilings can be constructed. The tiling information for each triangle t_i is stored as the pair of vertex indices from which arrows originate. These indices are 1, 2, or 3, corresponding to the first, second, or third vertex appearing in the list representing t_i. There are two arrows originating from vertex $t_i(o_2)$ and one from $t_i(o_1)$. A triangle for which the assignments have been made is labeled complete. The algorithm requires constructing the list of all triangles surrounding a surface vertex. This set forms a cell associated with the vertex.

For a given surface vertex, if none of the assignments have been made, all triangles in its cell can be oriented with a predetermined pattern (Fig. 8A, B). If a subset of the triangles has already been

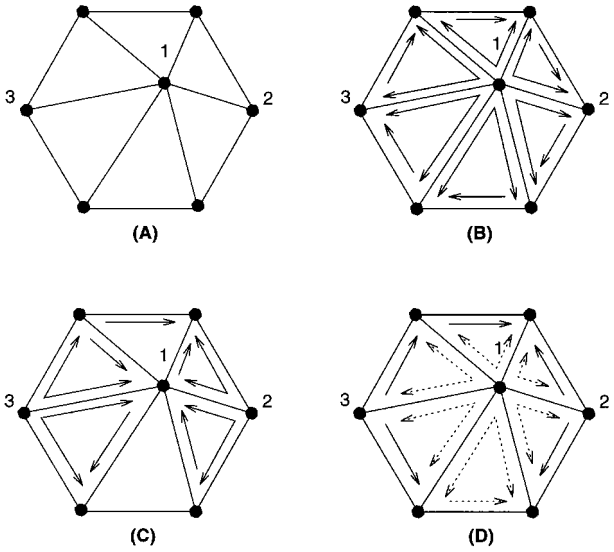

FIGURE 8. Consistent tiling assignment. (A) All triangles in cell 1 are unassigned. (B) Arbitrarily chosen tiling order. (C) Partial assignment obtained from earlier processing of cells surrounding vertices 2 and 3. (D) Completed assignment obtained by reassigning internal orientations (dotted arrows) according to (B). Other orientations are left unchanged.

assigned an orientation because of processing of neighbor cells occurring earlier in the list (Fig. 8C), the assignment can be completed by reorienting all the edges which are internal to the cell according to pattern in Figure 8B and leaving the cell boundaries unchanged (Fig. 8D).

Applications

The mesh generation system presented has already been used in a number of applications involving the solution to the Poisson equation in our laboratory[10] and is currently part of the Poisson–Boltzmann finite element solver we have developed (PBF). Here, we briefly describe the results obtained in applications of the algorithm to the generation of tetrahedral meshes for the alanine dipeptide and the BPTI protein. In each case, a probe radius of 1.4 Å was used to represent the effect of a water probe molecule. For the alanine dipeptide grids shown, an ionic radius of 2.0 Å was used to illustrate the positioning of the ion exclusion boundary in the mesh. In Figures 9 and 10 snapshots taken from the representation of meshes obtained for the alanine dipeptide are presented. All tetrahedral elements completely contained inside a 1-Å slice through the molecule are displayed, illustrating the resultant mesh structure. It is apparent that the triangulation algorithm applied to the grid-point distribution described earlier results in a mesh exhibiting the desired structure: high density of elements near the molecular surface; low density of elements elsewhere; and representation of the dielectric interface and the ion exclusion boundary as a set of triangular faces. The structure of the grid also ensures that the singularities located at the atomic centers (visible as the centers of the mesh subsets in the interior having a high degree of spherical symmetry) are always well separated from the dielectric interface, a property which cannot be guaranteed if a regular Cartesian grid is used without a high density of grid vertices. The structure of the surface mesh generated to calculate the induced surface charge distribution is visible in Figure 10 and illustrates how the algorithm achieves near volume-to-surface reduction in scaling behavior by concentrating most of the vertices and elements near the molecular surface. Although, in theory, the finite-element solution to the PB equation will have a complicated dependence on the mesh element aspect ratios, as well as on the degree of nonuniformity of the mesh, we have found in practice that once a sufficiently high mesh resolution was used in the neighborhood of the charges and the dielectric interface ($l_{sf} \simeq 16$), the solution remained fairly insensitive to the mesh density in the rest of the computational space.

The algorithm has been found to successfully generate adaptive tetrahedral meshes for all molecular geometries used (a database of 550 test cases was used as a test) in cases involving fewer than 200 atoms. For proteins, the algorithm can sometimes terminate before the mesh generation is complete because of ambiguous local representation of the interior/boundary near regions of high curvature on the molecular surface. Nevertheless, the method remains applicable in principle and has been used in the generation of meshes for small proteins. Figure 11 shows the molecular surface obtained from tetrahedral element faces for the BPTI protein. The statistics of the corresponding three-dimensional grid and mesh generated are shown in Table IV and indicate that the resulting grid-point distribution for the protein concen-

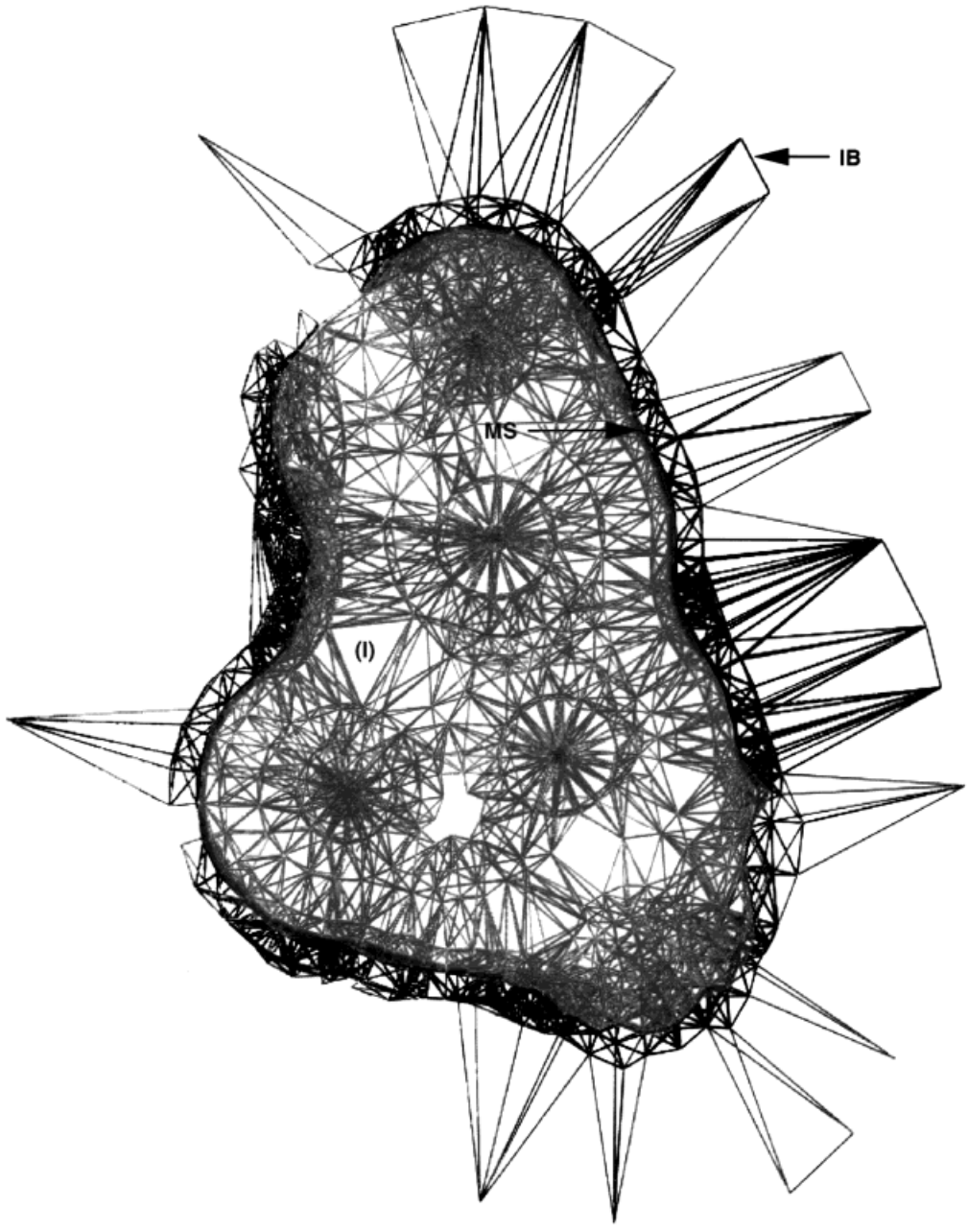

FIGURE 9. Representation of tetrahedral elements contained in a 1-Å slice through the mesh produced by the automated mesh generation system applied to the alanine dipeptide. The light and dark regions represent the interior (I) and exterior (E) of the fragment, respectively. The position of the Richards molecular surface (MS) and parts of the ion exclusion boundary (IB) are indicated. Grid generation parameters used include: $n_{sh} = 10$, $n_{sf} = 4$, $l_{sf} = 28$, $g_{d0} = 0.15$, $g_{d1} = 0.40$, $g_{d2} = 0.70$, $g_{db} = 12.00$, $\alpha_1 = 0.00$, $\alpha_2 = 0.70$.

trates almost 30% of the vertices in the Delaunay grid on the molecular surface itself. The CPU time on an IBM 580 RS6000 workstation required to construct the grid-point distribution was 0.8 s for the dipeptide and 38.9 s for the protein. In the latter case, most of the time is spent locating and trimming the cusps in the dielectric interface and work is presently underway to reduce the cost of this step. The corresponding triangulation times required to produce the full meshes were 15 s and

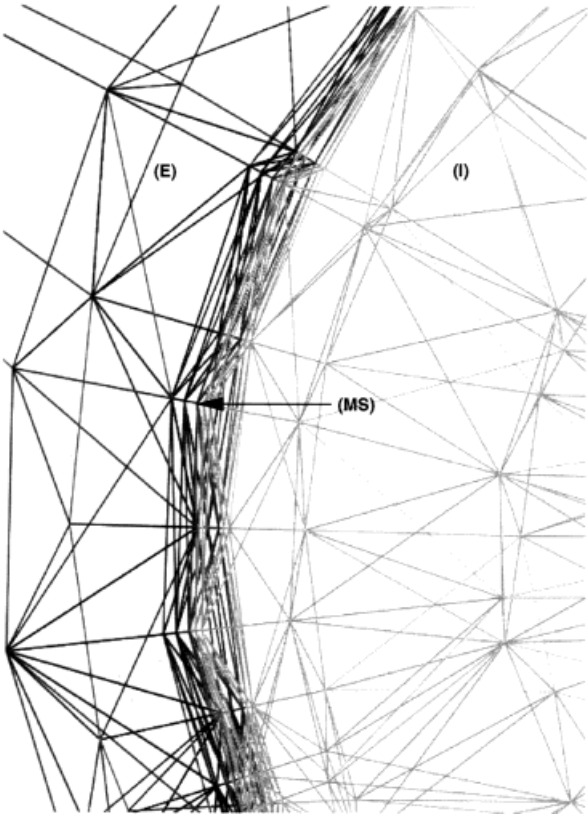

FIGURE 10. Detail of the mesh slice shown in Figure 9. The structure of the surface mesh constructed using the refinement stage of the triangulation process is visible.

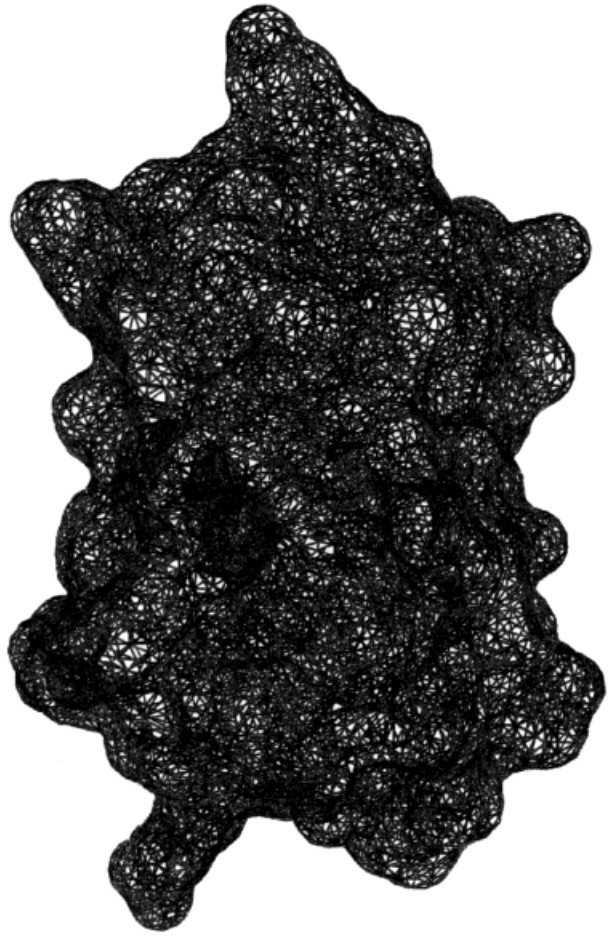

FIGURE 11. Representation of the Richards surface obtained as a subset of the three-dimensional mesh for the BPTI protein. Grid generation parameters used include: $n_{sh} = 8$, $n_{sf} = 4$, $l_{sf} = 18$, $g_{d0} = 0.15$, $g_{d1} = 0.40$, $g_{d2} = 0.00$, $g_{db} = 12.00$, $\alpha_1 = -0.50$, $a_2 = 0.70$.

49.1 s. Overall, the triangulation algorithm is capable of building the mesh at a rate of about 1400 points/s on this type of architecture.

The images were generated using the Iris Explorer^{TM} system[33] on a Silicon Graphics Indigo II workstation.

Conclusion

The algorithm we have presented enables the automatic generation of tetrahedral meshes for molecular geometries. The resulting meshes follow the constraints imposed by the presence of discontinuous parameter boundaries in the PB equation and will be used to compute finite element solutions to the equation in the following article. It should be noted that the strategy adopted for the construction of the triangulation is, to some extent, independent of the three-dimensional distribution of vertices themselves. Future work will involve the development of alternate algorithms for producing the grid-point distribution as part of the overall goal of further improving the efficiency of the method. The problem of internal cavity identification in proteins is also currently being investigated.

Acknowledgments

The authors thank Anthony Nicholls, Ranganathan Bharadwaj, Sundaram Sridharan, Andreas Windemuth, and Barry Honig for many instructive discussions regarding molecular surfaces and continuum electrostatic models as well as for

TABLE IV.
Statistics for Grid and Mesh Produced for the BPTI Protein. Triangulated Molecular Surface for the System Is Shown in Figure 11.

Grid type	Object	Number
Delaunay grid[a]	Atomic centers[c]	454
	Interior vertices	42029
	Surface vertices	11985
	Exterior vertices	1083
	Boundary vertices	150
	Total	55701
	Surface cusps	39
Complete grid[b]	Total	67686
Mesh	Tetrahedra	347878
	Surface triangles	23976

[a]Includes only vertices used in the constrained Delaunay triangulation.
[b]Includes vertices inserted in surface mesh with the direct meshing algorithm.
[c]A united atom model of the protein was used.

the use of the DelPhi program. C. M. C. thanks Timothy Baker for helpful discussions regarding the general problem of tetrahedral mesh generation. This work was supported through funding from NIH grant NIHGM40526. C. M. C. acknowledges support from the FCAR fund.

References

1. B. Honig and A. Nicholls, *Science*, **268,** 1144 (1995).
2. J. Warwicker and H. C. Watson, *J. Mol. Biol.*, **157,** 671 (1982).
3. I. Klapper, R. Hagstrom, R. Fine, K. Sharp, and B. Honig, *Proteins*, **1**, 47 (1986).
4. M. K. Gilson and B. Honig, *Nature*, **330,** 84 (1987).
5. A. Nicholls and B. Honig, *J. Comput. Chem.*, **4,** 435 (1991).
6. M. Holst and F. Saied, *J. Comput. Chem.*, **14,** 105 (1993).
7. R. Zauhar and R. J. Morgan, *J. Mol. Biol.*, **186,** 815 (1985).
8. R. Bharadwaj, A. Windemuth, A. Nicholls, S. Sridharan, and B. Honig, *J. Comput. Chem.*, **16,** 898 (1995).
9. Y. N. Vorobjev, J. A. Grant, and H. A. Scheraga, *J. Am. Chem. Soc.*, **114,** 3189 (1992).
10. C. Cortis, J. M. Langlois, M. D. Beachy, and R. A. Friesner, *J. Chem. Phys.* (in press).
11. A. A. Rashin and B. Honig, *J. Phys. Chem.*, **89,** 5588 (1985).
12. F. M. Richards, *Ann. Rev. Biophys. Bioeng.*, **6,** 151 (1977).
13. M. L. Connolly, *J. Appl. Cryst.*, **16,** 548 (1983).
14. V. I. Lebedev, *Zh. Vychisl. Mat. Mat. Fiz.*, **15,** 48 (1975).
15. V. I. Lebedev, *Sibirskii Matematicheskii Zhurnal*, **18,** 99 (1975).
16. T. You and D. Bashford, *J. Comput. Chem.*, **16,** 743 (1995).
17. M. L. Connolly, *J. Appl. Cryst.*, **18,** 499 (1985).
18. C. W. Mastin, J. F. Thompson, and Z. U. Z. Warsi, *Numerical Grid Generation Foundations and Applications*, North-Holland, Amsterdam, 1985.
19. L. R. Hermann, *J. Eng. Mech. Dic. ASCE*, **12,** 749 (1976).
20. O. C. Zienkiewicz and D. V. Phillips, *Int. J. Numer. Meth. Eng.*, **3,** 519 (1971).
21. R. Löhner and P. Parkikh, *Int. J. Numer. Meth. Fluids*, **8,** 1135 (1988).
22. R. Löhner, J. Camberos, and M. Merriam, *Comp. Meth. Appl. Mech. Eng.*, **95,** 343 (1992).
23. A. Nicholls, private communication (1993).
24. M. S. Shephard and M. K. Georges, *Int. J. Numer. Meth. Eng.*, **32,** 709 (1991).
25. G. Voronoi, *J. Reine Angew. Math.*, **134,** 198 (1908).
26. M. Tanemura, T. Ogawa, and N. Ogita, *J. Comput. Phys.*, **51,** 191 (1983).
27. N. Medvedeev, *J. Comput. Phys.*, **67,** 223 (1986).
28. S. Fortune, *Algorithmica*, **2,** 153 (1987).
29. R. E. M. Moore and I. O. Angell, *J. Comput. Phys.*, **105,** 301 (1993).
30. L. P. Chew, *Algorithmica*, **4,** 97 (1989).
31. A. Bowyer, *Comput. J.*, **24,** 162 (1981).
32. T. J. Baker, *Engineering with Computers*, **5,** 161 (1989).
33. *IRIS Explorer User's Guide*, Silicon Graphics Inc., 1992.